Feature Hashing

NYU Large Scale Learning Class

John Langford, Microsoft Resarch, NYC

March 24, 2014

Features in Practice: Engineered Features

Hand crafted features, built up iteratively over time, each new feature fixing a discovered problem.
In essence, boosting where humans function as the weak learner.

1. +Good understanding of what's happening.
2. +Never fail to learn the obvious.
3. +Small RAM usage.
4. -Slow at test time. Intuitive features for humans can be hard
5. -Low Capacity. A poor fit for large datasets. (Boosted) Decision trees are a good compensation on smaller datasets.
6. -High persontime.

Features in Practice: Learned Features

Use a nonlinear/nonconvex possibly deep learning algorithm.

1. +Good results in Speech & Vision.
2. +Fast at test time.
3. +High capacity. Useful on large datasets.
4. -Slow training. Days to weeks are common.
5. -Wizardry may be required.

Features in Practice: Count Features

An example: for each (word, ad) pair keep track of empirical expectation of click $\hat{E}[c|(word, ad)]$.

1. +High capacity.
2. +Fast learning. Counting is easy on map-reduce architectures.
3. +fast test time. Lookup some numbers, then compute an easy prediction.
4. -High RAM usage. Irrelevant features take RAM.
5. -Correlation effects lost. Adding explicit conjunction features takes even more RAM.

Features in Practice: sparse words

Generate a feature for every word, ngram, skipgram, pair of (ad word, query word), etc... and use high dimensional representation.

1. +High capacity.
2. +Correlation effects nailed.
3. +fast test time. Lookup some numbers, then compute an easy prediction.
4. -Slow learning Linear faster than decision tree, but parallel is tricky.
5. -High RAM usage

Features in Practice: sparse words

Generate a feature for every word, ngram, skipgram, pair of (ad word, query word), etc... and use high dimensional representation.

1. +High capacity.
2. +Correlation effects nailed.
3. +fast test time. Lookup some numbers, then compute an easy prediction. This lecture.
4. -Slow learning Linear faster than decision tree, but parallel is tricky. This lecture + Allreduce lecture.
5. -High RAM usage This lecture.

What is hashing?

Hash function: string $\rightarrow \{0, 1\}^b$
A hash function maps any string into a range seemingly at random.

What is hashing?

Hash function: string $\rightarrow \{0,1\}^b$
A hash function maps any string into a range seemingly at random.

Hash table = Hash function + Array< Pair<string, int> > of length $\{0,1\}^b$

What is hashing?

Hash function: string $\rightarrow \{0,1\}^b$
A hash function maps any string into a range seemingly at random.

Hash table $=$ Hash function $+$ Array$<$ Pair$<$string, int$>$ $>$ of length $\{0,1\}^b$

Perfect hash $=$ overfit decision tree mapping n fixed (and known in advance) strings to integers $\{1,n\}$.

How does feature address parameter?

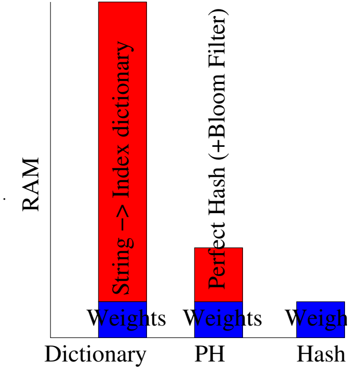
1. Hash Table (aka Dictionary): Store hash function + Every string + Index.

How does feature address parameter?

1. Hash Table (aka Dictionary): Store hash function + Every string + Index.
2. Perfect Hash (+Bloom Filter): Store Custom Hash function (+ bit array).

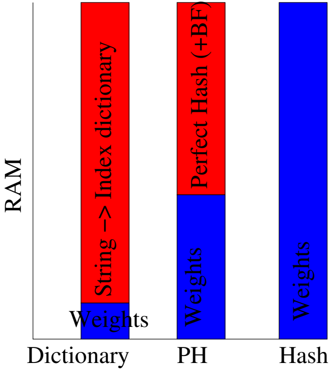
How does feature address parameter?

1. Hash Table (aka Dictionary): Store hash function + Every string + Index.
2. Perfect Hash (+Bloom Filter): Store Custom Hash function (+ bit array).
3. Hash function: Store Hash function.



How does feature address parameter?

1. Hash Table (aka Dictionary): Store hash function + Every string + Index.
2. Perfect Hash (+Bloom Filter): Store Custom Hash function (+ bit array).
3. Hash function: Store Hash function.



More weights is better!

Objection: Collisions!

Valid sometimes: particularly with low dimensional hand engineered features.

Objection: Collisions!

Valid sometimes: particularly with low dimensional hand engineered features.

Theorem: If a feature is duplicated $O(\log n)$ times when there are $O(n)$ features, at least one version of the feature is uncollided when hashing with $\log(n \log n)$ bits.
Proof: Similar to Bloom filter proof.

Example 1: CCAT RCV1

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw
-1 | econom stock rate month year invest week produc report govern pric index million shar end reserv foreign research inflat gdp growth export consum output annual industr cent exchang project trad fisc servic base compar prev money bank debt balanc gold daily import agricultur ago estimat ton prelimin deficit currenc nation

...

Example 1: CCAT RCV1

1 | tuesday year million short compan vehicl line stat financ commit exchang plan corp subsid credit issu debt pay gold bureau prelimin refin billion telephon time draw
-1 | econom stock rate month year invest week produc report govern pric index million shar end reserv foreign research inflat gdp growth export consum output annual industr cent exchang project trad fisc servic base compar prev money bank debt balanc gold daily import agricultur ago estimat ton prelimin deficit currenc nation
...

Run:
vw -b 24 --loss_function logistic --ngram 2 --skips 4 -c rcv1.train.raw.txt --binary
to see progressive validation loss 4.5%: about 0.6% better than linear on base features.

Objection: Incomprehensible!

Objection: Incomprehensible!

Use –audit to decode. Or, keep your own dictionary on the side if desirable.

vw-varinfo rcv1.test.raw.txt.gz = perl script in VW distribution for automatically decoding and inspecting results.

Use of Hash: Feature Pairing

Once you accept a hash function, certain operations become very easy.
-q df pairs every feature in namespaces beginning with d with every feature in namespaces beginning with f.
But how?

Use of Hash: Feature Pairing

Once you accept a hash function, certain operations become very easy.
-q df pairs every feature in namespaces beginning with d with every feature in namespaces beginning with f.
But how?

Feature = (index,weight)
pair_weight = d_weight * f_weight
pair_index = (d_index * magic + f_index) & mask
This is done *inline* for speed.

Use of Hash: Ngrams

2gram = a feature for every pair of adjacent words.
3gram = a feature for every triple od adjacent words, etc...
ngram = ...

Features computed in the same fashion as for -q

(More clever solution = rolling hash, not yet implemented.)

Computed by the parser on the fly (since #features/example only grows linearly).

Learning Reductions

In many applications, you must have multiple predictors. Hashing allows all these to be mapped into the same array using a different offsets saving gobs of RAM and programming headaches.

–oaa, –ect, –csoaa, and others.

Example 2: Mass Personalized Spam Filtering

1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique tokens.

How do we construct a spam filter which is personalized, yet uses global information?

Example 2: Mass Personalized Spam Filtering

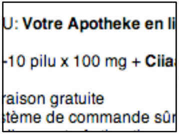
1. $3.2 * 10^6$ labeled emails.
2. 433167 users.
3. $\sim 40 * 10^6$ unique tokens.

How do we construct a spam filter which is personalized, yet uses global information?

Bad answer: Construct a global filter + 433167 personalized filters using a conventional hashmap to specify features. This might require $433167 * 40 * 10^6 * 4 \sim$ 70Terabytes of RAM.

Using Hashing

Use hashing to predict according to: $\langle w, \phi(x) \rangle + \langle w, \phi_u(x) \rangle$



text document (email)

2x

$\mathbf{x}$

NEU
Votre
Apotheke
en
ligne
Euro
...

$+$

$\mathbf{x}_l$

USER123_NEU
USER123_Votre
USER123_Apotheke
USER123_en
USER123_ligne
USER123_Euro
...

tokenized, duplicated bag of words

h

$\mathbf{x}_h$

323
0
5235
0
0
123
0
626
232
...

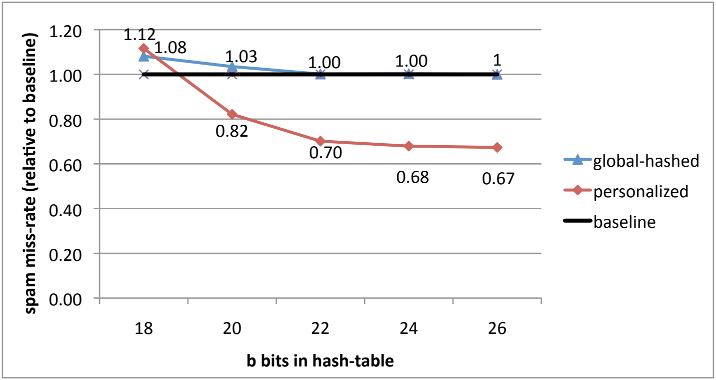
hashed, sparse vector

$\mathbf{w}^\top \mathbf{x}_h$

classification

(in VW: specify the userid as a feature and use -q)

Results



2^{26} parameters $= 64$M parameters $= 256$MB of RAM.
An x270K savings in RAM requirements.

Features sometimes collide, which is scary, but then you love it

Generate a feature for every word, ngram, skipgram, pair of (ad word, query word), etc... and use high dimensional representation.

1. +High capacity.
2. +Correlation effects nailed.
3. +Fast test time. Compute an easy prediction.
4. +Fast Learning (with Online + parallel techniques. See talks.)
5. +/-Variable RAM usage. Highly problem dependent but fully controlled.

Another cool observation: Online learning + Hashing = learning algorithm with fully controlled memory footprint $\Rightarrow$ Robustness.

References, prequels

1. Reinforcement Learning: An Introduction, Richard S. Sutton and Andrew G. Barto, MIT Press, Cambridge, MA, 1998. Chapter 8.3.1 hashes states.
2. CRM114 http://crm114.sourceforge.net/, 2002. Uses hashing of grams for spam detection.
3. Apparently used by others as well, internally.
4. Many use hashtables which store the original item or a 64+ bit hash of the original item.

References, “modern” hashing trick

1. 2007, Langford, Li, Strehl, Vowpal Wabbit released.
2. 2008, Ganchev & Dredze, ACL workshop: A hash function is as good as a hashmap empirically.
3. 2008/2009, VW Reimplementation/Reimagination/Integration in Stream (James Patterson & Alex Smola) and Torch (Jason Weston, Olivier Chapelle, Kilian).
4. 2009, AIStat Qinfeng Shi et al, Hash kernel definition, Asymptopia Redundancy analysis
5. 2009, ICML Kilian et al, Unbiased Hash Kernel, Length Deviation Bound, Mass Personalization Example and Multiuse Bound.